Support Vector Machine

**Short note about Support Vector Machine**:

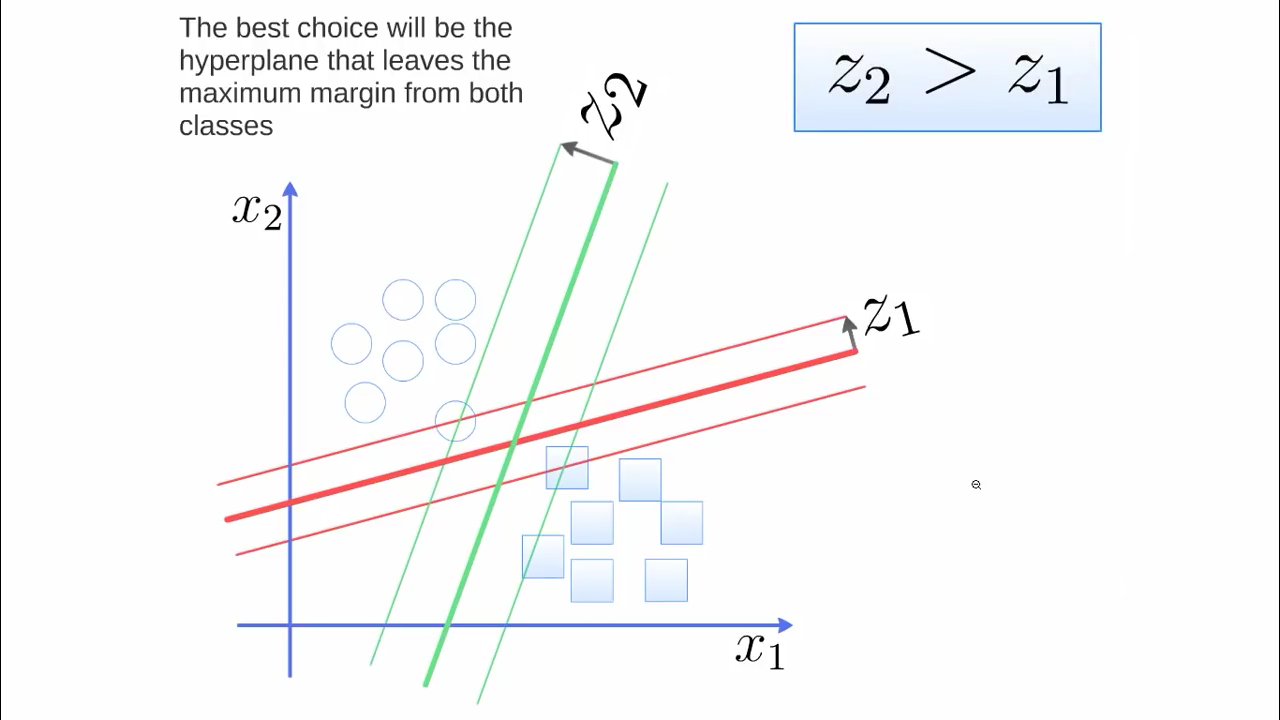
Support Vector Machine(SVM) is a supervised machine learning algorithm which can be used for both classification or Regression problems. Usually used for classification.

In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot).

**HyperPlane-**

Given 2 or more labeled classes of data,it acts as a discriminative classifier,which is known as optimal hyperplane that seperates all the classes.

Hyperplane is a linear decision surface that splits the space into two parts.



* **Identify the right hyper-plane (Scenario-1):** Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.



You need to remember a thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better”. In this scenario, hyper-plane “B” has excellently performed this job.

**Identify the right hyper-plane (Scenario-2):** Here, we have three hyper-planes (A, B and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?



Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**. Let’s look at the below snapshot:



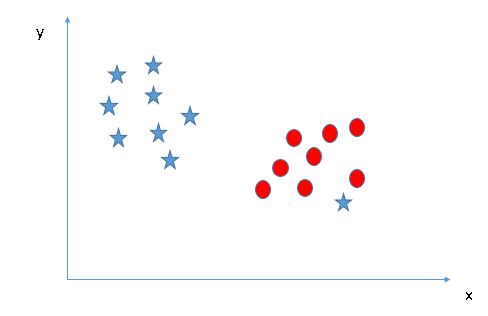
Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

**Identify the right hyper-plane (Scenario-3):**Hint:Use the rules as discussed in previous section to identify the right hyper-plane

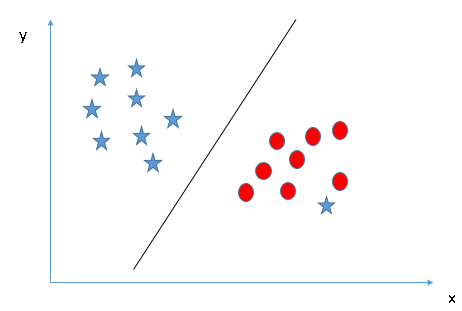


Some of you may have selected the hyper-plane **B** as it has higher margin compared to **A.** But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**

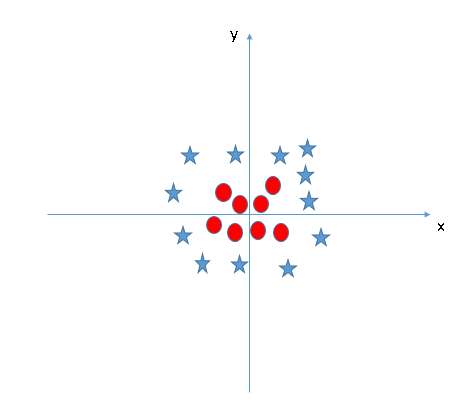
**Can we classify two classes (Scenario-4)?:** Below, I am unable to segregate the two classes using a straight line, as one of star lies in the territory of other(circle) class as an outlier. As I have already mentioned, one star at other end is like an outlier for star class. SVM has a feature to ignore outliers and find the hyper-plane that has maximum margin. Hence, we can say, SVM is robust to outliers.

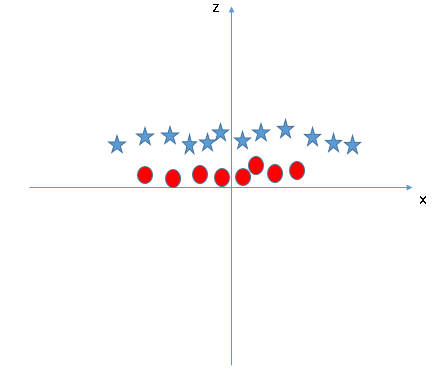


As I have already mentioned, one star at other end is like an outlier for star class. SVM has a feature to ignore outliers and find the hyper-plane that has maximum margin. Hence, we can say, SVM is robust to outliers.



**Find the hyper-plane to segregate to classes (Scenario-5):** In the scenario below, we can’t have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.



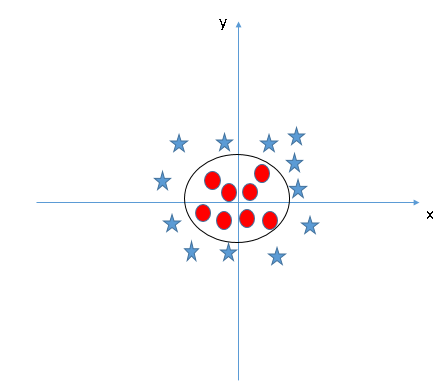
SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature z=x^2+y^2. Now, let’s plot the data points on axis x and z:

In above plot, points to consider are:

* All values for z would be positive always because z is the squared sum of both x and y
* In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.

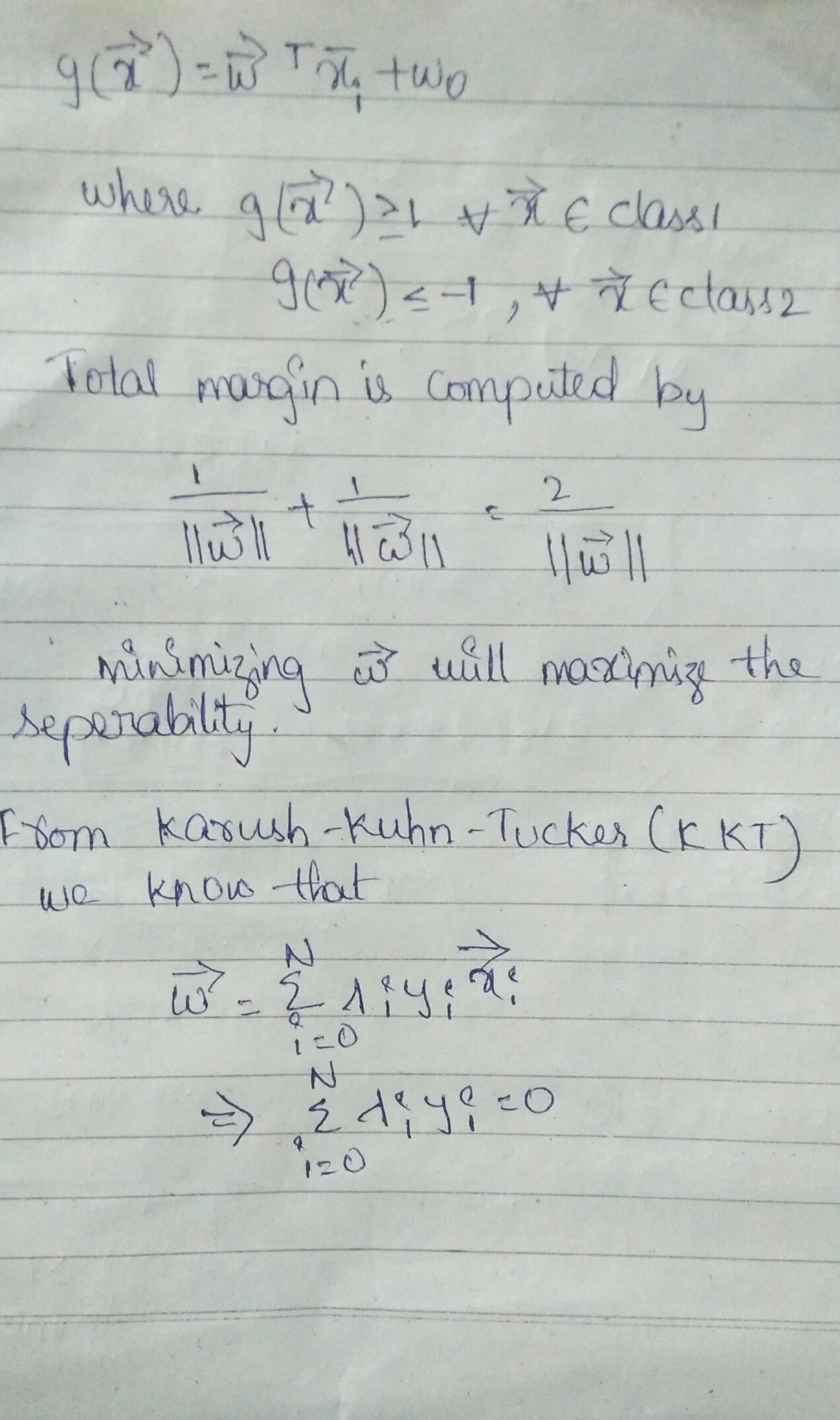
In SVM, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, SVM has a technique called the kernel trick. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs you’ve defined.

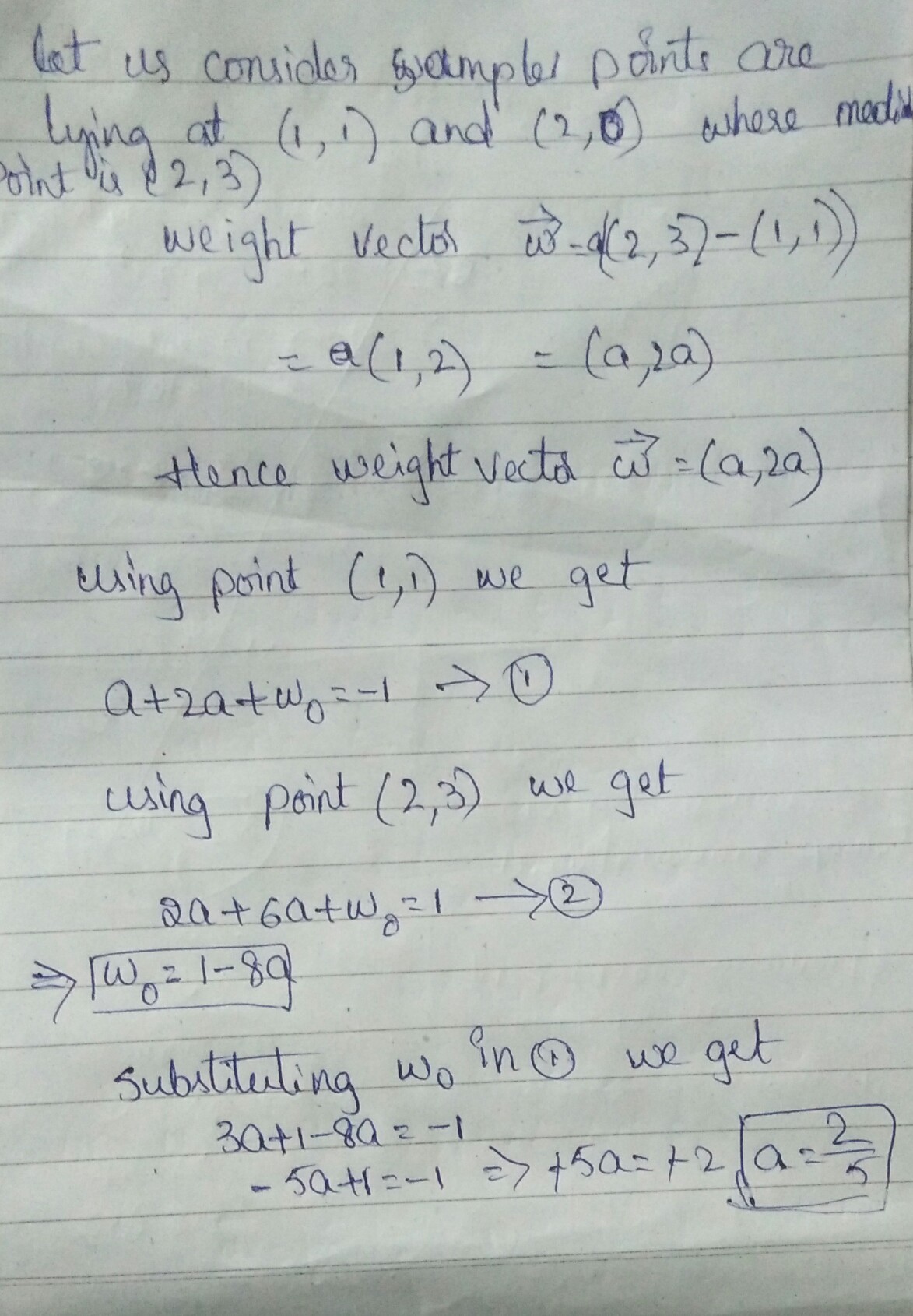
When we look at the hyper-plane in original input space it looks like a circle:

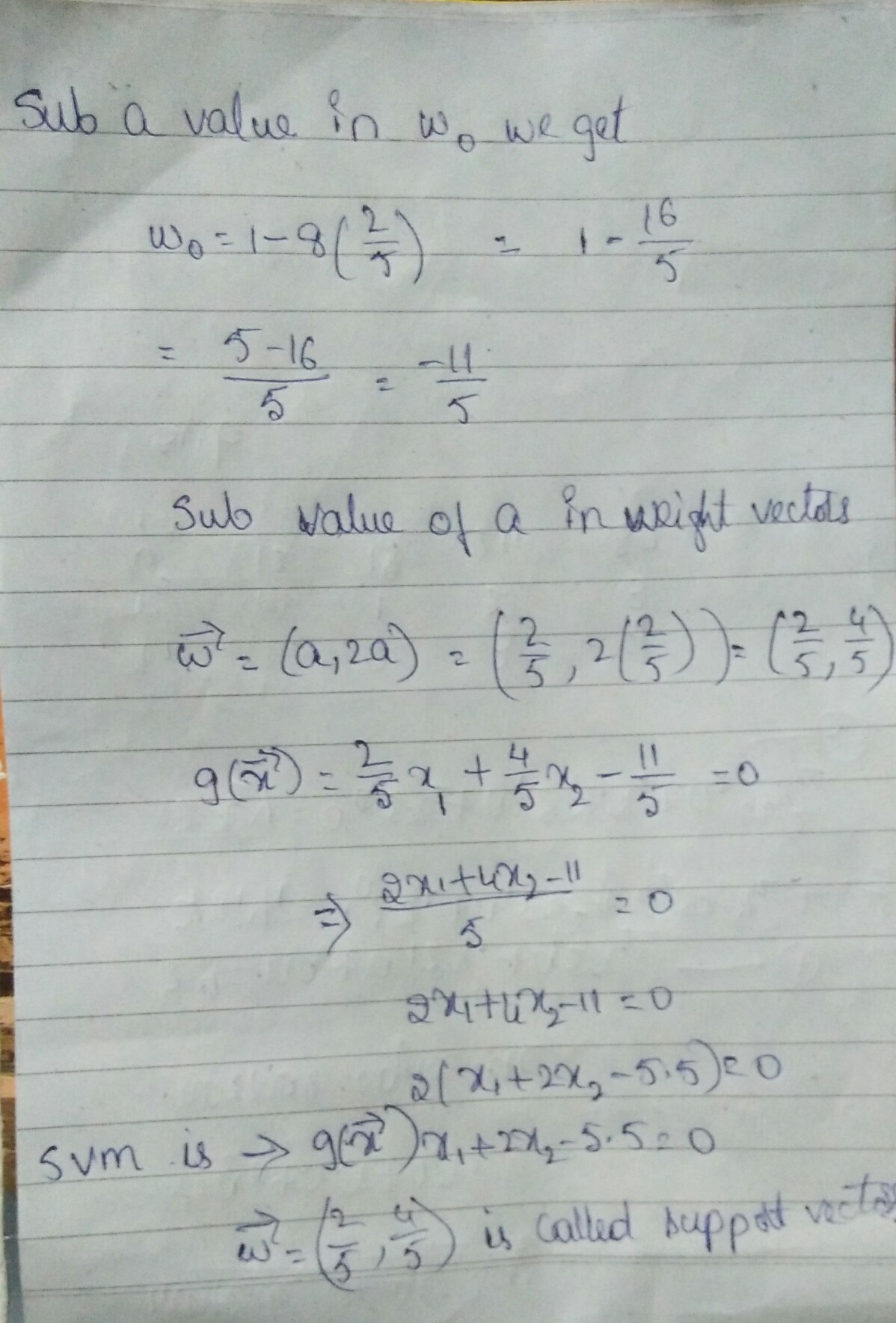


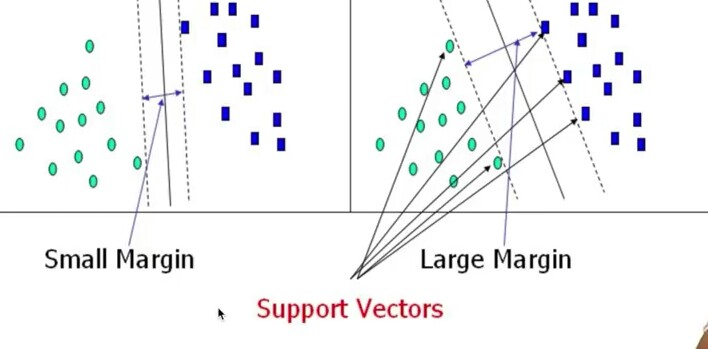
Decode Complex Algorithm

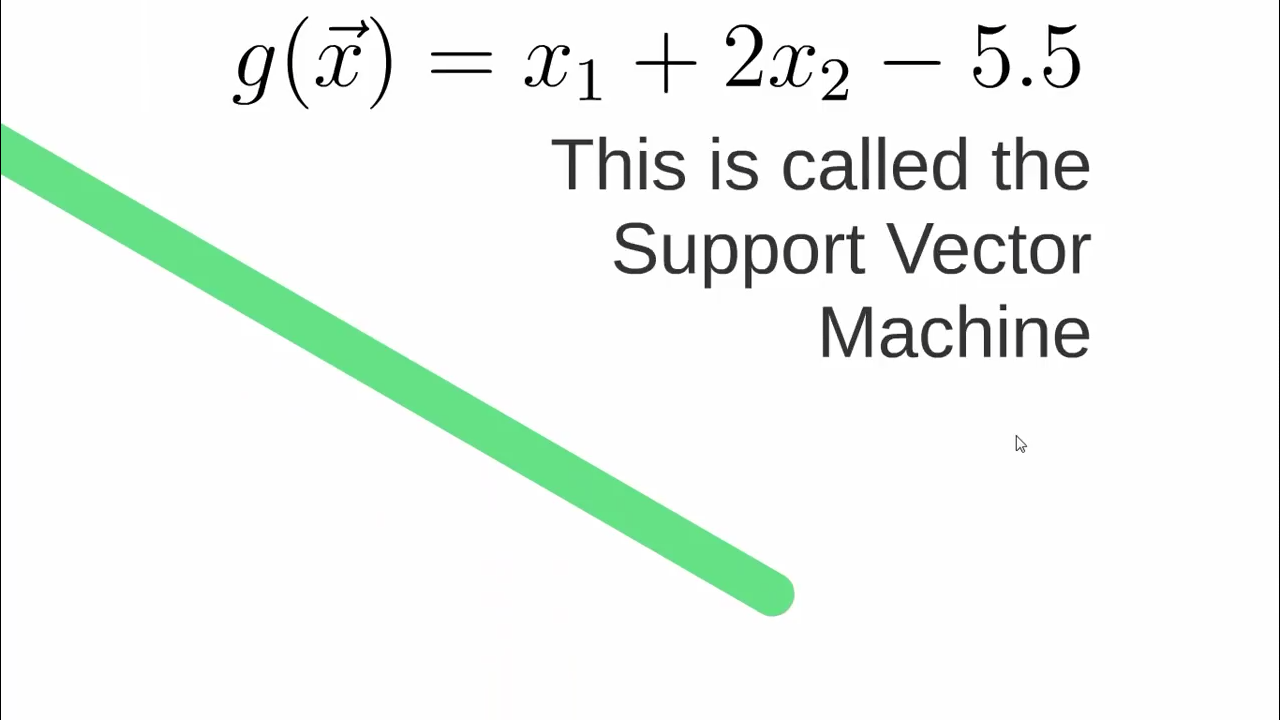
Step: 1











USE CASES

* Classify data according to categories
* Find out who set for the job and who doesn’t using different parameters
* Clearly defining or recognizing the best fit line.
* Eg. Hand Written digit classification and general recognizion
* -Determines the ideal separation of two entities
* • Face detection – SVM classify parts of the image as a face and non-face and create a square boundary around the face.
* • Text and hypertext categorization – SVMs allow Text and hypertext categorization for both inductive and transudative models. They use training data to classify documents into different categories. It categorizes on the basis of the score generated and then compares with the threshold value.
* • Classification of images – Use of SVMs provides better search accuracy for image classification. It provides better accuracy in comparison to the traditional query-based searching techniques.
* • Bioinformatics – It includes protein classification and cancer classification. We use SVM for identifying the classification of genes, patients on the basis of genes and other biological problems.
* • Protein fold and remote homology detection – Apply SVM algorithms for protein remote homology detection.
* • Handwriting recognition – We use SVMs to recognize hand written characters used widely.
* • Generalized predictive control(GPC) – Use SVM based GPC to control chaotic dynamics with useful parameters.

Python Code without library

PYTHON CODE

import matplotlib.pyplot as plt

from matplotlib import style

import numpy as np

style.use('ggplot')

class Support\_Vector\_Machine:

def \_\_init\_\_(self, visualization=True):

self.visualization = visualization

self.colors = {1:'r',-1:'b'}

if self.visualization:

self.fig = plt.figure()

self.ax = self.fig.add\_subplot(1,1,1)

# train

def fit(self, data):

self.data = data

# { ||w||: [w,b] }

opt\_dict = {}

transforms = [[1,1],

[-1,1],

[-1,-1],

[1,-1]]

all\_data = []

for yi in self.data:

for featureset in self.data[yi]:

for feature in featureset:

all\_data.append(feature)

self.max\_feature\_value = max(all\_data)

self.min\_feature\_value = min(all\_data)

all\_data = None

# support vectors yi(xi.w+b) = 1

step\_sizes = [self.max\_feature\_value \* 0.1,

self.max\_feature\_value \* 0.01,

# point of expense:

self.max\_feature\_value \* 0.001,

]

# extremely expensive

b\_range\_multiple = 2

# we dont need to take as small of steps

# with b as we do w

b\_multiple = 5

latest\_optimum = self.max\_feature\_value\*10

for step in step\_sizes:

w = np.array([latest\_optimum,latest\_optimum])

# we can do this because convex

optimized = False

while not optimized:

for b in np.arange(-1\*(self.max\_feature\_value\*b\_range\_multiple),

self.max\_feature\_value\*b\_range\_multiple,

step\*b\_multiple):

for transformation in transforms:

w\_t = w\*transformation

found\_option = True

# weakest link in the SVM fundamentally

# SMO attempts to fix this a bit

# yi(xi.w+b) >= 1

#

# #### add a break here later..

for i in self.data:

for xi in self.data[i]:

yi=i

if not yi\*(np.dot(w\_t,xi)+b) >= 1:

found\_option = False

#print(xi,':',yi\*(np.dot(w\_t,xi)+b))

if found\_option:

opt\_dict[np.linalg.norm(w\_t)] = [w\_t,b]

if w[0] < 0:

optimized = True

print('Optimized a step.')

else:

w = w - step

norms = sorted([n for n in opt\_dict])

#||w|| : [w,b]

opt\_choice = opt\_dict[norms[0]]

self.w = opt\_choice[0]

self.b = opt\_choice[1]

latest\_optimum = opt\_choice[0][0]+step\*2

for i in self.data:

for xi in self.data[i]:

yi=i

print(xi,':',yi\*(np.dot(self.w,xi)+self.b))

def predict(self,features):

# sign( x.w+b )

classification = np.sign(np.dot(np.array(features),self.w)+self.b)

if classification !=0 and self.visualization:

self.ax.scatter(features[0], features[1], s=200, marker='\*', c=self.colors[classification])

return classification

def visualize(self):

[[self.ax.scatter(x[0],x[1],s=100,color=self.colors[i]) for x in data\_dict[i]] for i in data\_dict]

# hyperplane = x.w+b

# v = x.w+b

# psv = 1

# nsv = -1

# dec = 0

def hyperplane(x,w,b,v):

return (-w[0]\*x-b+v) / w[1]

datarange = (self.min\_feature\_value\*0.9,self.max\_feature\_value\*1.1)

hyp\_x\_min = datarange[0]

hyp\_x\_max = datarange[1]

# (w.x+b) = 1

# positive support vector hyperplane

psv1 = hyperplane(hyp\_x\_min, self.w, self.b, 1)

psv2 = hyperplane(hyp\_x\_max, self.w, self.b, 1)

self.ax.plot([hyp\_x\_min,hyp\_x\_max],[psv1,psv2], 'k')

# (w.x+b) = -1

# negative support vector hyperplane

nsv1 = hyperplane(hyp\_x\_min, self.w, self.b, -1)

nsv2 = hyperplane(hyp\_x\_max, self.w, self.b, -1)

self.ax.plot([hyp\_x\_min,hyp\_x\_max],[nsv1,nsv2], 'k')

# (w.x+b) = 0

# positive support vector hyperplane

db1 = hyperplane(hyp\_x\_min, self.w, self.b, 0)

db2 = hyperplane(hyp\_x\_max, self.w, self.b, 0)

self.ax.plot([hyp\_x\_min,hyp\_x\_max],[db1,db2], 'y--')

plt.show()

data\_dict = {-1:np.array([[1,7],

[2,8],

[3,8],]),

1:np.array([[5,1],

[6,-1],

[7,3],])}

svm = Support\_Vector\_Machine()

svm.fit(data=data\_dict)

predict\_us = [[0,10],

[1,3],

[3,4],

[3,5],

[5,5],

[5,6],

[6,-5],

[5,8]]

for p in predict\_us:

svm.predict(p)

svm.visualize()

output:

[1 7] : 1.271999999999435

[2 8] : 1.271999999999435

[3 8] : 1.0399999999995864

[5 1] : 1.0479999999990506

[ 6 -1] : 1.7439999999985962

[7 3] : 1.0479999999990506

Graph:

